# Theory of Computation

**Prof. Raghunath Tewari**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Kanpur**

## Lecture – 04
## Computation by DFA and Regular operation

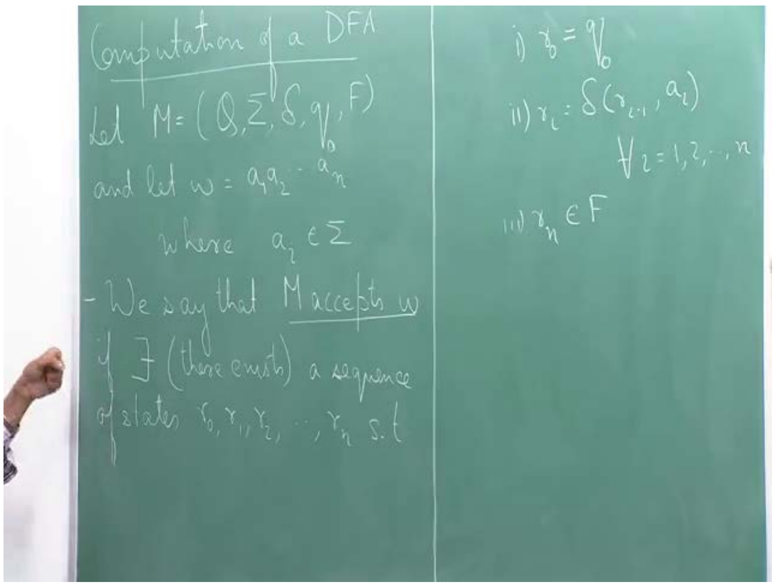
Welcome to the 4th lecture of this course. In this lecture, we will formally define what it means to say that a DFA computes a string. And we will also look at operations on languages known as regular operations.

(Refer Slide Time: 00:14)

Lecture 4: Computation by a DFA
Regular Operations

We will start by looking at the formal definition of computation of a DFA. We gave an informal meaning of what does it mean for a DFA to compute a string in an earlier lecture. Today, we are going to look at a formal definition. We look at couple of examples after that and then finally, we will end by studying some properties of the languages that are accepted by deterministic finite automaton. To being let us look at the formal definition of what does it mean for a DFA to compute a string.

(Refer Slide Time: 01:15)



Computation of a DFA

Let $M = (Q, \Sigma, \delta, q_0, F)$

and let $w = a_1 a_2 \cdots a_n$

where $a_i \in \Sigma$

- We say that M accepts w

if $\exists$ (there exists) a sequence of states $r_0, r_1, r_2, \cdots, r_n$ s.t

i) $r_0 = q_0$

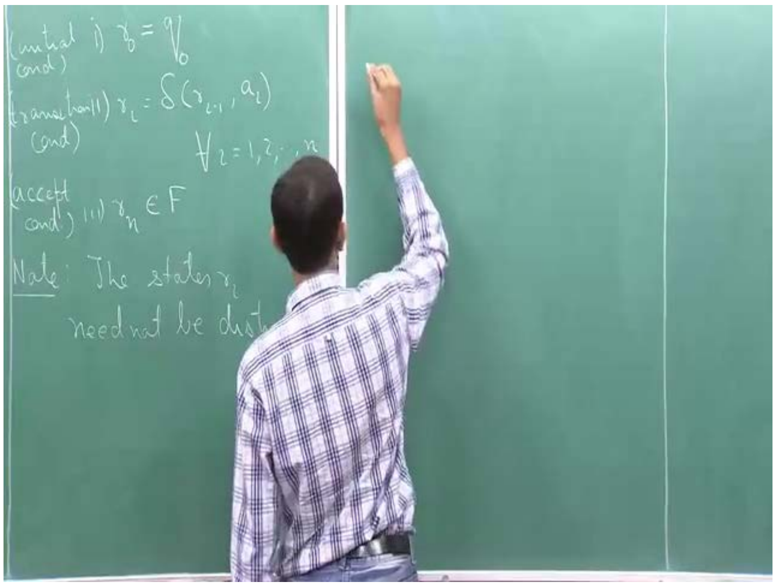
ii) $r_i = \delta(r_{i-1}, a_i)$ $\forall i = 1, 2, \cdots, n$

iii) $r_n \in F$

Let M be a DFA. So, M equals Q, sigma, delta, q naught and F. Let me reminds, so here Q is the set of state, sigma is the alphabet, delta is the transition function, q naught is the start state, and F is the set of accept state. And let w be a string over sigma. So, w equals let us say it as n symbol, so w equal to a 1, a 2 so on till a n where a i is a symbol in sigma. We say that M accepts w. If there exists, so this is the short hand notation for there exist, I will be using this frequently in this course.

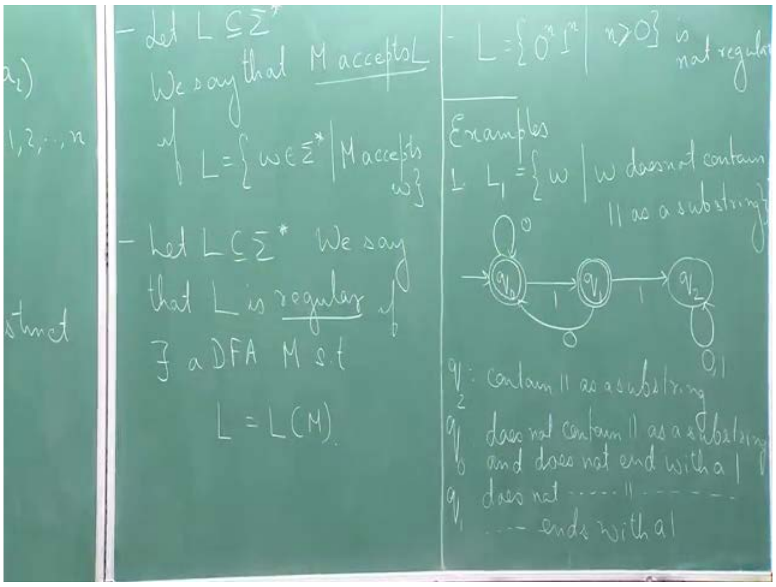
So, if there exist let me write it in bracket for now, a sequence of states r 0, r 1, r 2 up to r n such that three properties hold. Firstly, r 0 is the start state; for all i equal to 1, 2 up to n. And the third is the acceptance condition that is r n belongs to F. So, this is just formal way of saying what does it mean to say that a DFA M accepts a string w. So, DFA M accepts a string w.

(Refer Slide Time: 04:27)


(initial cond) i) $r_0 = q_0$
(transition cond) ii) $r_i = \delta(r_{i-1}, a_i)$
$\forall i = 1, 2, \ldots, n$
(accept cond) iii) $r_n \in F$
Note: The states $r_i$ need not be dist

If there is a sequence of states and note here that we write this that the states r i need not be distinct. If there is a sequence of states coming from my set q, such that the first state is the start state. And for every i going from 1 through n, r i is the state that we get by applying the transition function to r i minus 1 and a i. Basically, r 1 will be delta r 0 comma a 1; r 2 will be delta r 1 comma a 2. And finally, r n will be delta r n minus 1 comma a n. So, this is basically the transition condition. Let me write this here. So, this is the initial condition, this is the transition condition, and this is the acceptance condition. And the final state, r n is going to be is a state that belongs to F. So, this is what we mean to say us the DFA accepts a string w.

(Refer Slide Time: 06:27)


$a_2)$
$1, 2, \ldots, n$
stinct

- Let $L \subseteq \Sigma^*$
We say that M accepts L
if $L = \{w \in \Sigma^* \mid M \text{ accepts } w\}$

- Let $L \subseteq \Sigma^*$ We say that L is regular if $\exists$ a DFA M s.t.
$L = L(M)$

- $L = \{0^n 1^n \mid n \geq 0\}$ is not regular

Examples
1 $L_1 = \{w \mid w \text{ does not contain } 11 \text{ as a substring}\}$

$$\begin{tikzcd} q_0 \arrow[loop above, "0"] \arrow[r, "1"] & q_1 \arrow[l, bend left, "0"] \arrow[r, "1"] & q_2 \arrow[loop below, "0,1"] \end{tikzcd}$$

$q_2$: contains 11 as a substring
$q_0$: does not contain 11 as a substring and does not end with a 1
$q_1$: does not ..... 11 ..... ends with a 1

Now using this, what do we mean when we say that a DFA accepts a language. Let L be a language over the alphabet sigma. We say that M accepts L, if L equals the set of all strings such that M accepts w. And once again, I should emphasis here that this equal sign is very important. This means that if I look at every string that belongs to this set on the right hand side, so every string that is accepted by M, it belongs to the language L; and whatever string belongs to the language L, it is accepted by M. So, it is exactly equal one is not a proper subset of the other so that is very important.

And the third is a notation for such type of languages. Once again, let L be a language. We say that L is regular, if there exist a DFA M such that L equals L of M. So, recall from last time, this notation means, so this is another way of saying the language, which is accepted by the DFA M. So, this is just a notation for this language of strings that are accepted by M. If there is some DFA that accepts a language L, we say that language is regular. Now, the natural question that arises is of course, we have seen several examples of DFA in our last lecture and today, we will see a couple of more.

But more importantly the question that arises is that are there languages which are not regular, is it true that all languages that we have a regular, which means that for every language there is a DFA which accepts it or are there also certain languages which are not-regular and not only so can we prove them that they are not regular. So, mathematically can we show that there exist languages for which there is a no DFA that we can construct which accepts that language. And as it turns out the answer to that questions is yes, there are languages which are not-regular.

Let me just state this at this point. For example, the language let me call it L which consists of all string of the font 0 to the power n, 1 to the power n, where n is greater than or equal to 0 is not regular. This consists of all strings which have a sequence of zeros followed by a sequence of ones, such that the number of zeros is equal to the number of ones. So, we will not prove this right now, so we will in later lecture, we will come back to this language. This is going to be very important language, we come back to this language and we will prove why this language is not regular, but for the time being, I just want to state this.

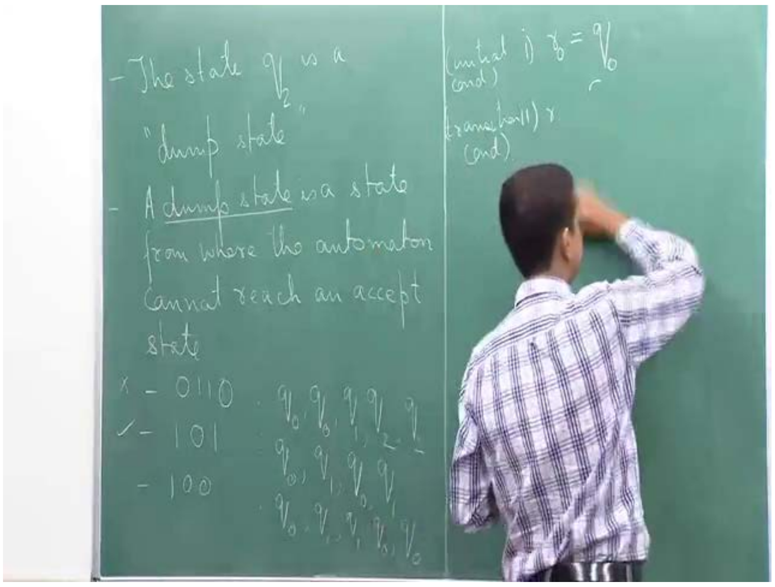
Now let us look at couple of more example of DFA, and then we will move to some properties. So, the first example that we will see is let us call it L 1 set of strings w such that w does not contain 1 1 as a substring. So, last time, we saw an example, where we considered strings which contain a certain substring as part of itself. All strings that contained a particular substring. Today, we look at a language which consists of all strings that do not contain 1 1.

How do we construct a DFA for this? So, the idea is going to be as follows. So, we will try to scan the string. So, when we are reading the string, we try to scan it; and the moment we find a 1 once present anywhere in the string we will go to what we call a dump state. So, we will go to a state from where we cannot actually come out, the DFA cannot escape and come to an accept state any further.

Let us look at the construction. We start at a state q 0. So, if I see single 1, I go to a state q 1; and if I see two successive 1s, I go to this state q 2. And now what do these state means. So, q 2 will correspond to the state where I have seen two successive 1s, two ones which are next to one another. If I reach q 2, I am going to stay at q 2, no matter whether I see as 0 or 1 after this. q 0 is going to correspond to a state where I have not seen a single 1; and q 1 is going to correspond to a state where I have seen just a single 1. Let me write it here. So, q 2 is corresponds to the set of all strings that contain 1 1 as a substring; q 0 is does not contain 1 1 as a substring, and it does not end with a 1; and q 1 is going to be that it does not contain 1 1 as a substring and it ends with a 1.

So, from q 0, if I see a 1, I go to q 1; from q 0, if I see a 0, I will stay at q 0. From q 1, if I see a 1, it means that I have encounter to consecutive ones which takes me to q 2. And if I see a 0 from q 1, it means that the string that I have seen so far does not end with a 1 which means I should go back to q 0. So, this is automaton. And what are the accept states of this automaton. So, I want to accepts all string that does not contain 1 1, so it will have two accept states one is q 0 and the other is q 1.

(Refer Slide Time: 16:00)



- The state $q_2$ is a "dump state"
- A dump state is a state from where the automaton cannot reach an accept state

x - 0110 . $q_0, q_0, q_1, q_2, q_2$

✓ - 101 $q_0, q_1, q_0, q_1$

- 100 . $q_0, q_1, q_1, q_0, q_0$

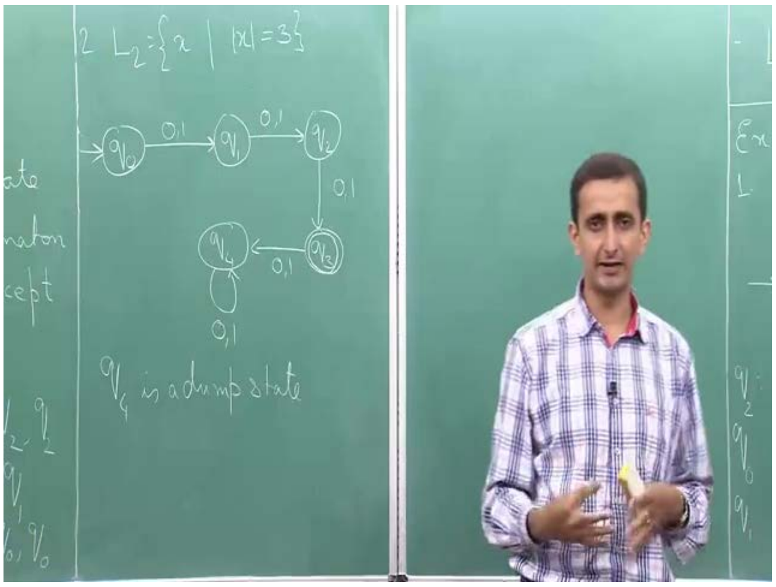
(initial i) $q_0 = q_0$
cond)

(transition) x
(cond)

In this example, the point that I want to emphasis is that the state q 2 is a what we call a dump state. In other words, formally if I want to define it, a dump state is a state from where the automaton cannot reach an accept state. If I look at a couple of examples of strings to this automaton earlier, so if I look at let say a string 0 1 1 0, sorry 0 1 1 0 what would be the computation of this string on this automata. The automaton starts at q 0; from q 0 on a 0, it states at q 0; then again from q 0 on a 1, it goes to q 1. On seeing the second one, it will go to q 2; and then on seeing the last 0 it stays at q 2. This string makes the automaton end up at q 2, which is not an accept state, hence this string is not accepted and that should be the case because it does contain 1 1 as a substring.

On the other hand, if you look at an example like 1 0 1, but would the computation, so on 1, the automaton from q 0 it will go to q 1. On seeing a 0 after 1, it will go back to q 0; and then again on seeing the next one, it will go to q 1, because q 1 is an accept state this string is accepted. Similarly, if you have a string that ends with 0, let say 1 0 0 you can see that from q 0 it goes q 1 then from q 1 it will go to q 0 and then it will stay at q 0. So, this is a string which contains 1 1 as a substring this is string which does not contain 1 1 as a substring, but ends with a 1. And this is a string which does not contain 1 1 as a substring and does not end with a 1.

(Refer Slide Time: 19:33)

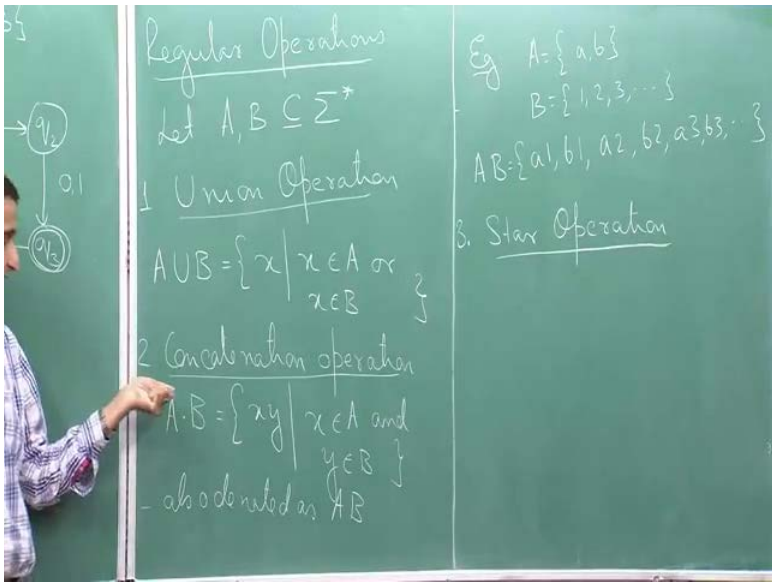


Let us look another simple example. Let us look at the set of all strings x such that the length of x equals let say 3. So, I want to accept all those strings whose length is 3. It is not difficult, so we start at a state q 0. Now on seeing the first character, we actually do not care what the characters are, the characters that we see whether it is a 1 or a 0. So, on seeing the first character, I will just mark it with the 0 1, which means that it does not matter whether its 0 or 1, I go to a state q 1. On seeing the next character, I go to q 2; and then on seeing the third character, I go to a state q 3; and then after that no matter what character I see, I will stay at I will go to a state q 4, and I am going to stay thereafter. So, whether I see a 0, or a 1, I will not move.

So, what is my accept state or accept states in this case so this state q 0 corresponds to all those strings that that have a length 0, which is actually only the empty string epsilon; q 1 corresponds to those strings that have length 1, q 2 corresponds to those strings that have length 2 and q 3 corresponds to those strings that have length 3. So, this is going to be my accept state. And, q 4 will correspond to all those strings that have length 4 or more this is a simple. So, again in this case if you note the state q 4 is a dump state, because from q 4, I cannot reach an accept state.

Now, we will look at what are called regular operations. So, there are a certain operations, there are some operations that we can apply to languages that give us some other languages. So, they can either be binary operations or they can be unary operations. So, binary operation or languages basically takes two languages, and produces a third language; and a unary operation basically takes one language, and produces another language.

(Refer Slide Time: 22:43)



Regular Operations

Let $A, B \subseteq \Sigma^*$

1 Union Operation

$A \cup B = \{ x \mid x \in A \text{ or } x \in B \}$

2 Concatenation operation

$A \cdot B = \{ xy \mid x \in A \text{ and } y \in B \}$

- also denoted as $AB$

Eg $A = \{a, b\}$

$B = \{1, 2, 3, \cdots\}$

$AB = \{a1, b1, a2, b2, a3, b3, \cdots\}$

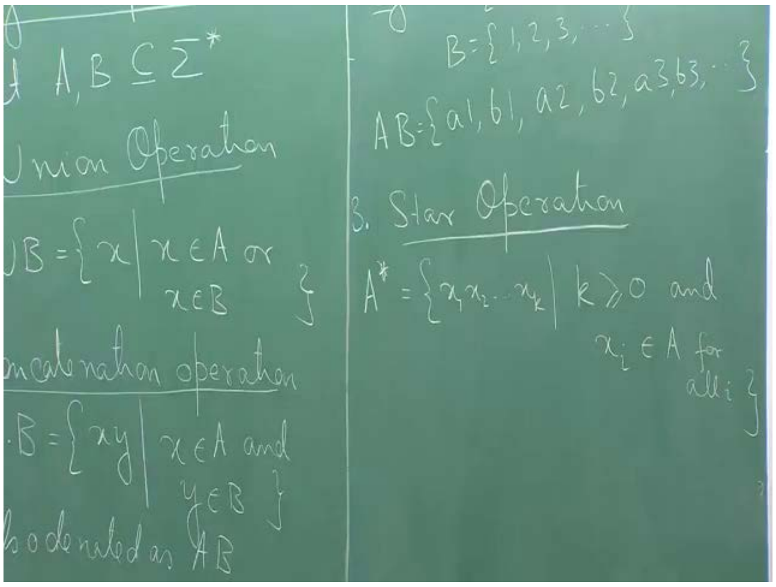
3. Star Operation

The first regular operations that we will see are called the Union Operation. Actually, these operations are something that you have seen even in your middle school, but in the context of languages nevertheless we will define it. Before I define let me just mention one thing. Let A comma B be two languages. They can be languages over any alphabet sigma. So, we define A union B in the natural way. So, A union B consists of all strings x such that x either belongs to the language A or x belongs to the language B.

The next operation, so this is something that we have seen, the next is what is called the concatenation operation. So, we denote this with the dot in the middle. So, A dot B is the set of all strings of the form x y such that x belongs to A, and y belongs to B. So, this operation is the concatenation operation, where I take two strings x and y, and I append one to the end of the other string. Here, x y means that x comes first and then y is appended to the end of x. This operation of concatenation is often also denoted as just A B. So, we just drop this dot in the middle and we just write it as AB.

As a small example let us look at this. So, suppose our set A has elements a and b, and B has let say 1, 2, 3 and so on. Then what is A B. So, A B will consist of all strings of the form. First we will have a 1, b 1, then a 2, b 2, a 3, b 3 and so on, this is an infinite language sorry this is an infinite language which consists of all those strings that we get by taking one string from A and one string from B and concatenating then ok. So, they need not just be. Here, I have taken a and b as languages which have strings consisting of a single symbol, but in general they can have more than one symbol as well. And the final operation that we are going to define today is what is called the star operation. So, the first two operations that is union and concatenation, these were binary operations; in other words, these were operations which took as input two languages and produced one language as output.

(Refer Slide Time: 27:16)



$A, B \subseteq \Sigma^*$

Union Operation

$B = \{ x \mid x \in A$ or $x \in B \}$

ncatenation operation

$.B = \{ xy \mid x \in A$ and $y \in B \}$

so denoted as AB

$B = \{1, 2, 3, \cdots\}$

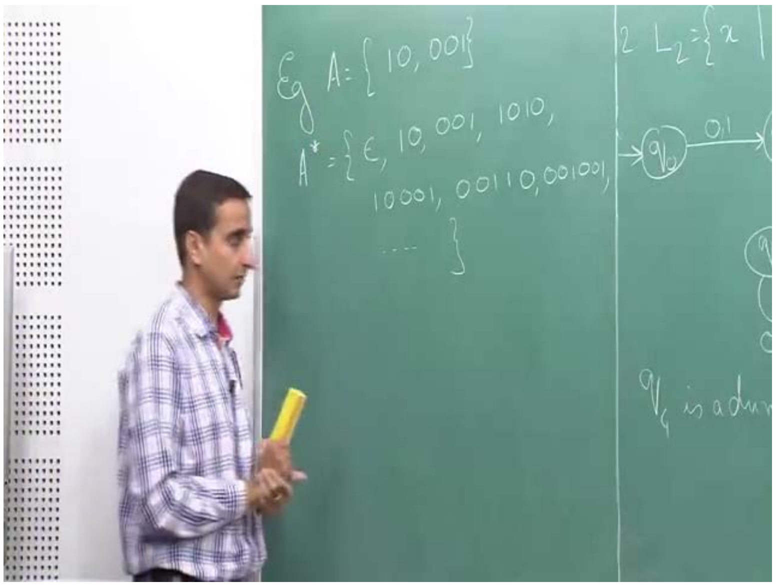
$AB = \{a1, b1, a2, b2, a3, b3, \cdots\}$

3. Star Operation

$A^* = \{ x_1 x_2 \cdots x_k \mid k \geq 0$ and $x_i \in A$ for all $i \}$

The star operation is a unary operation; it just operates on one language. We define a star as the set of all strings of the form some x 1, x 2 sorry x 1 concatenated with x 2 up to some x k, where k is greater than or equal to 0. And each x i belongs to A for all i. So, basically I take some k strings and k need not be a fixed number, k can be any number that is greater than or equal to 0. I take k strings and I concatenate them and I just keep on building.

(Refer Slide Time: 28:23)



Let us look at an example. Let A be the language which consists of the strings 1 0 and let say 0 0 1. Then what is A star? So in A star first I look at all strings that I get by taking k equal to 0. If k is 0, the only string that is possible is epsilon. Then I take k equal to 1, so I look at all strings from a by combining only one string at a time. So, it consists only of the string 1 0, and 0 0 1. Next I take k equal to 2, so I combine two strings. If A has two strings the way I can combine these two is in four different ways.

So, I will get 1 0 1 0, if I take 1 0 and 1 0 or I can have 1 0 0 0 1. So, if I take the first string with the second string. The third I can get is 0 0 1 1 0. If I take this first and then I apply the second string or I can get 0 0 1 0 0 1. And similarly, I can go to three strings which are concatenations of 3 strings 4 strings and so on from here. In particular, this language A star is an infinite language, if A is some non-trivial language. So, I will stop here.